Master's Thesis

Multiverse: Mobility pattern understanding improves localization accuracy

Beknazar Abdikamalov

Department of Computer Science and Engineering

Graduate School of UNIST

2019

Multiverse: Mobility pattern understanding improves localization accuracy

Beknazar Abdikamalov

Department of Computer Science and Engineering

Graduate School of UNIST

Abstract

This paper presents the design and implementation of Multiverse, a practical indoor localization system that can be deployed on top of already existing WiFi infrastructure. Although the existing WiFi-based positioning techniques achieve acceptable accuracy levels, we find that existing solutions are not practical for use in buildings due to a requirement of installing sophisticated access point (AP) hardware or special application on client devices to aid the system with extra information. Multiverse achieves sub-room precision estimates, while utilizing only received signal strength indication (RSSI) readings available to most of today's buildings through their installed APs, along with the assumption that most users would walk at the normal speed. This level of simplicity would promote ubiquity of indoor localization in the era of smartphones.

Contents

List of Figures

I Introduction

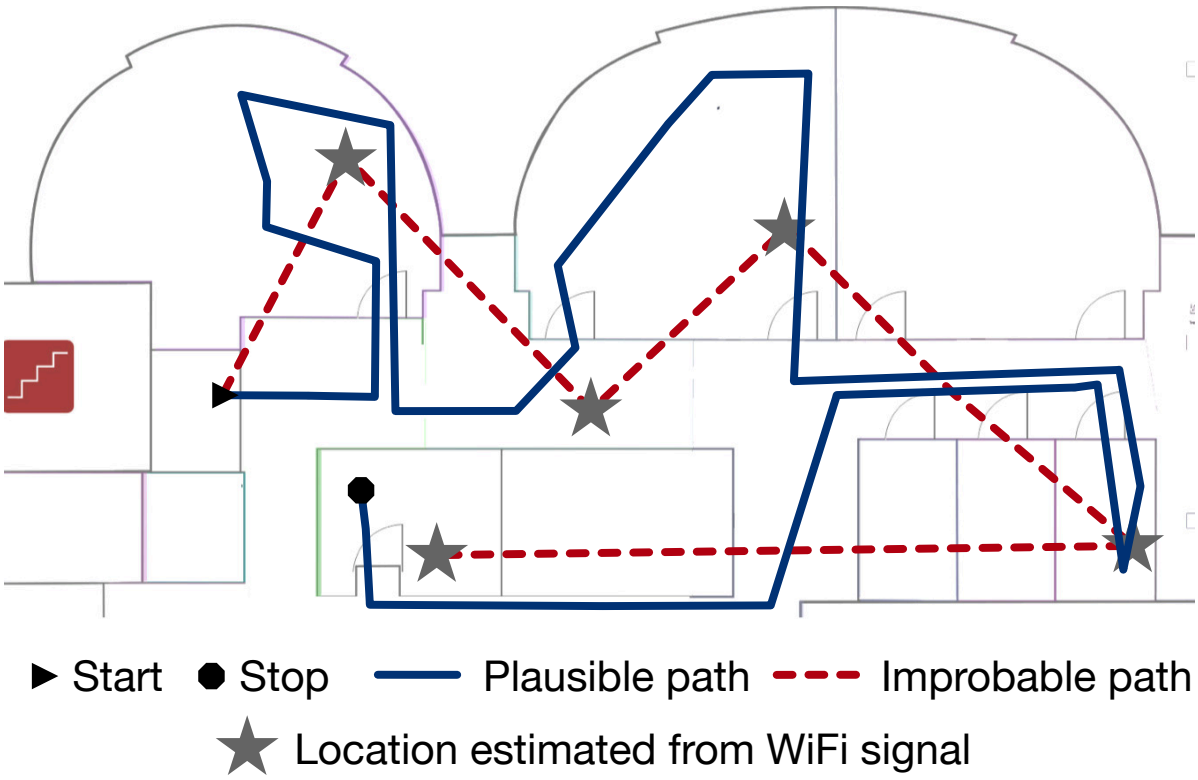


Figure 1: Illustration of the main idea

Indoor localization is considered unarguably one of the most important enabling techniques for mobile services. It has drawn significant amount of interest from research communities during the last few decades. There have been several directions of research but, inspired by the GPS (global positioning system), a major portion of researchers have given their focuses on developing a hand-held system that exploits some sort of signals received by the system to pinpoint an indoor position. The variety of the signals utilized for this purpose subsumes WiFi [1], Bluetooth [2,3], FM radio [4,5], RFID (radio-frequency identification) [6–9], ultrasound or sound [10, 11], light [12, 13], and magnetic field [14, 15]. Thanks to all the efforts, it is now widely accepted that when the signals are properly fingerprinted per location, the accuracy of indoor positioning can be as much as few meters. According to a recent study exploiting much more detailed low-level signal information including [16, 17], the accuracy can be even more improved to a sub-meter scale.

However, it is disappointing that none of these achievements became a de-facto standard of indoor localization and the majority of billions of smartphone users in the world are still not benefited from an indoor localization technique in their living spaces. To address the practicality issues hindering adoption of such methods in reality, we propose a new immediately deployable indoor localization system, *Multiverse*. Unlike previously available methods, Multiverse keeps away from the following two nuisances: 1) user device engagement and 2) labor-intensive pro-

cedure. The necessity of user device engagement for some reasons such as signal measurement, signal analysis, dead reckoning, and map matching makes user devices battery-hungry and sluggish. Therefore, it is not surprising that a set of methods that essentially asks for the user device engagement is under-appreciated. Especially, a branch of methods continuously burdening a user device to have an estimate of the position such as dead reckoning is not welcomed. Similarly, the necessity of any labor intensive procedure such as signal fingerprinting at a wide range of positions in an indoor structure makes a localization method shunned. Crowdsourcing is considered to be a good alternative to such a procedure, but if it demands more than natural behaviors of an average person, it is also unpromising. To our knowledge, available indoor localization techniques with room-level or better accuracy are not free from either or both of these nuisances.

Multiverse escapes from those problems by implementing a localization algorithm running in the backend server of a WiFi infrastructure, which deduces the most plausible mobile trajectory over multiple imperfect location estimates made from WiFi signals captured in the infrastructure. By doing so, it removes its dependency on user devices and raises its localization accuracy to a room-level. At the same time, it removes the necessity of a labor-intensive procedure of signal fingerprinting by roughly crowdsourcing the positions on a floor plan from randomly chosen users in the indoor structure. The conjecture behind Multiverse is that the deduced plausible trajectory with consideration of indoor walking speed substantially reduces the uncertainty involved in spot-wise location estimates. We empirically validate the conjecture by implementing Multiverse in the WiFi infrastructures of several indoor places and by obtaining its accuracy of 1.6 meters improved from 5.1 meters observed in a infrastructure-based spot-wise localization method. The idea is illustrated in Figure 1, where aiming to construct plausible paths would improve overall localization accuracy.

We implemented Multiverse using the RSSI data feed coming from the Cisco Wireless LAN controller, which is designed to send real-time information about clients devices that are heard by access points of the network. We run evaluations against one of the best-performing infrastructure-based system RADAR-inspired solution [18,19] under the same circumstances; we describe the testbed in VI. Our experiments show that Multiverse achieves a median localization error of 1.6 m, and the 80^{th} percentile error is 4.1 m.

II Related work

In the literature of indoor localization, numerous solutions have been proposed in the past two decades. Generally, they fall into 2 classes: RSS based, modeling based, and fusion techniques.

Firstly, RSS based approaches, which rely on RSSI readings from the target device at multiple APs and combines them via triangulation along with a propagation model to locate the devices [1,19–26]. These approaches have an advantage of being readily deployable, but largely based on laptops with quite different antenna forms (antenna polarization) or need to read RSSI values directly from the devices. Second, angle of arrival (AoA) based approach, which calculates the AoAs of the multipath signals received at each AP, finds the direct path to the target device and then applies triangulation to localize [16, 17, 27–31]. The best AoA based approaches showed a great level of precision on the order of 0.4 m [16, 17]. However, these methods are relatively challenging to deploy, since they require additional hardware changes by introducing as high as 8 antennas [16, 31], or new boxes itself with rotating antennas [17], or require special APs to access IQ samples [16, 29].

There are other recently proposed methods that use device sensors such as gyroscopes, accelerometers, etc., along with RSS values [32–37]. However, it would not be practical in the sense that clients are required to install special applications or allow certain modifications in their operating systems. However, we want to continuously localize a mobile device with only these existing WiFi signals without any additional infrastructure, as well as without requiring access to the device readings directly.

III System overview

Main challenges and design goals. Multiverse builds its components around three major design goals: *(i)* The system should be usable in buildings using existing their WiFi infrastructure. *(ii)* Continuously and accurately track users with any smartphone connected to APs, without directly accessing their devices. *(iii)* The system should output location traces which are humanly possible, without violating human walking speed.

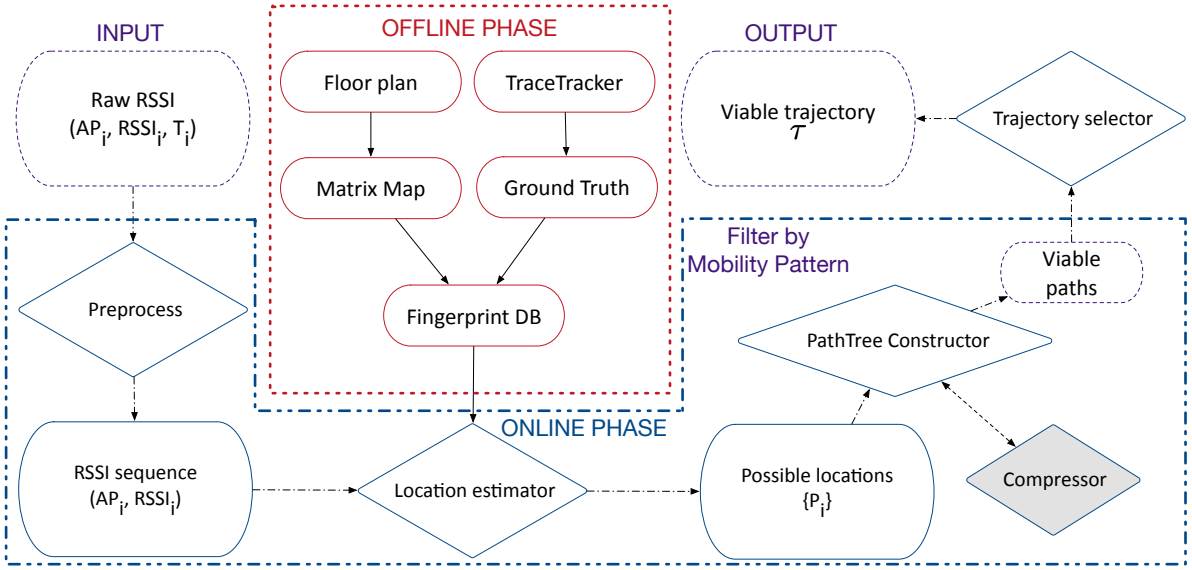


Figure 2: Multiverse system workflow.

System workflow. We divide the system stem into two main categories: offline and online phases. Figure 2 illustrates their workflow. Offline phase actions refer to processes that are to prepare the system and are done only once in the beginning, while online phase tasks are done while tracking users.

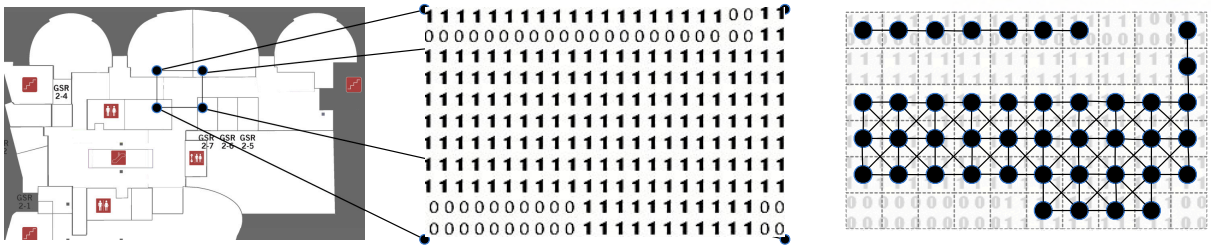
A floor plan is available in almost any building and shows the structure of the floors from above, including the relationships between rooms, spaces, and other physical features. Simply taking a direct line between two points in a floor plan is not necessary to be the walking distance or route between them due to the block of walls and other obstacles. Hence, we perform the following steps to get the real distances and paths between any locations in the floor by considering all the physical constraints. First, we convert the true-color image of the floor plan map to the gray-scale intensity image using MATLAB's Image Processing Toolbox [38]. Second, a binary matrix is created based on the gray-scale image by replacing all values above a determined threshold with 1s and setting all other values to 0s. In the evaluations, we determined the threshold using Otsu's method [39], which chooses the threshold value to minimize the intraclass variance of the thresholded black and white pixels. Lastly, the binary matrix is equally divided into a mesh of grids and we refer to each center of those grids as *node*. Length l of a grid can be 1-3 meters according to the general performance of fingerprinting-based localization

methods. In our experiment, we set $l = 1m$. By calculating the distances between all pairs of sample points, we have the distance matrix $D = [d_{ij}]$, where d_{ij} is the shortest distance calculated using Dijkstra's algorithm between two points p_i and p_j in the floor plan. This algorithm is described in more detail in the Algorithm 1.

To estimate location from received RSSI values it is necessary to have an RSSI to node mapping. To accomplish the mapping, we have developed an Android application $TraceTracker$, which can be used simply by pinpointing the locations on the screen while walking as shown in Figure 6. Using the data, we build a matrix M which maps each historical $AP \times RSSI$ to a vector of corresponding locations (nodes) $\vec{P}$.

Based on our experiments, we have found out that signal strength signals received at APs need to be preprocessed by performing the following steps: *(i)* Merge the possible nodes of signals with the same timestamp *(ii)* Skip signal data if impossible to connect with its temporally neighbouring signal data, if all of the connections between consecutive location nodes exceed the maximum walking speed of an average human.

After cleaning up the noisy or violating signal points, the system utilizes the fingerprint database M to get the list of possible locations for each received signal strength data using the location estimator described in the Algorithm 2. Then, a tree of all the possible paths is constructed by connecting the consecutive list of locations. Since, to avoid exponential growth of the tree, the tree is compressed periodically at certain time intervals to keep the tree at manageable size. Finally, the list of viable paths will be connected to get the most viable trajectory.



a) Floor plan b) Binary matrix c) Graph of nodes

Figure 3: Transform process of the algorithm 1

IV Offline phase

4.1 Converting floor plan image to a graph of nodes

Our goal in this phase is to convert a typical floor plan image into a binary matrix M, where each cell c of the matrix is defined as follows

$$M(c) = \begin{cases} 1, & \text{if } c \text{ is accessible} \\ 0, & \text{if } c \text{ is inaccessible} \end{cases}$$

To obtain the matrix above, we run the algorithm 1 on the obtained image file of a floor plan. Overall, this conversion gives us the following benefits:

1. Environmental constraints such as walls and doors are naturally considered to efficiently avoid during path construction processes. Constraint areas are considered as inaccessible areas.

2. Shortest path-finding algorithms can run efficiently. We are able to utilize Dijkstra's shortest path calculation between any points in the accessible space by treating the whole space as a graph.

3. Received signal strength to location converting models are able to map RSSI values into discrete locations, which improves memory and computational utilization.

A floor plan is available in almost any building and shows the structure of the floors from above, including the relationships among rooms, spaces, and other physical features. Simply taking a direct line between two points in a floor plan is usually not the walking route or distance between them due to the block of walls and other obstacles. Hence, we perform the following steps to get the real distances and paths between any 2 locations in the floor by considering all the physical constraints. First, we convert the true-color image of the floor plan map to the gray-scale intensity image using MATLAB's Image Processing Toolbox [38]. Second, a binary matrix is created based on the gray-scale image by replacing all values above a determined threshold with 1s and setting all other values to 0s. In the evaluations, we determined the threshold using Otsu's method [39], which chooses the threshold value to minimize the intraclass variance of the

thresholded black and white pixels. Lastly, the binary matrix is equally divided into a mesh of grids and we refer to each center of those grids as *node*. The length l of a grid can be 1-3 meters according to the general performance of fingerprinting-based localization methods. In our experiment, we set $l = 1m$.

An example of the converted floor plan can be seen in Figure 3. Dots in the accessible area refer to nodes which is the result if equally dividing the space into a mesh of grids.

Algorithm 1 Converting floor plan to graph of nodes

Input: Image of the floor plan

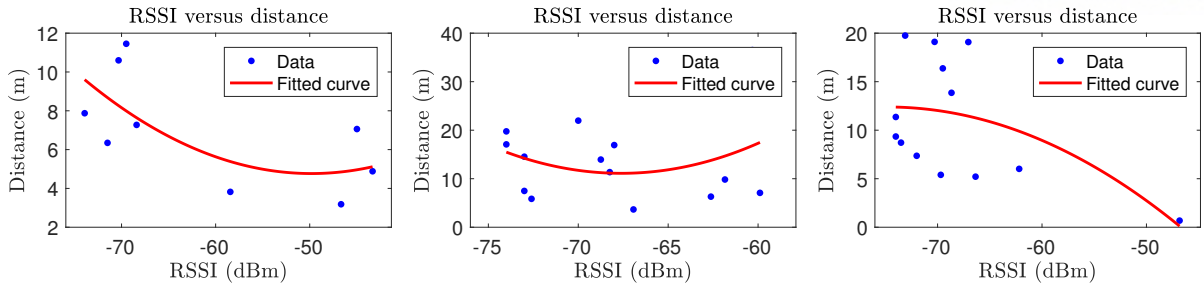
Output: Matrix map

1: Convert plan into a gray-scale image using the function $rgb2gray$;
2: Computer threshold of the gray image of the plan using Otsu's method;
3: Binarize the gray image using the determined threshold to get a binary matrix;
4: Uniformly divide the binary matrix into a mesh of grids;
5: Create a graph using the centers of the grids;

Also, after converting the floor plan to a graph of nodes, we do one more step which helps us to obtain realistic routes and distances while running the main algorithm. Since, simply taking a direct line between points in the floor plan will not give us the real walking routes between them due to the block of walls and other obstacles. We utilize the Floyd-Warshall algorithm [40] to determine the shortest paths between any nodes in the graph. The computation is run once only and stored in the matrix to quickly returning realistic routes and distances in the later stages of the system. This step will output the matrix $R = [r_{ij}]$, where r_{ij} is the shortest path between two points p_i and p_j in the floor plan.

Table 1: List of attributes of each WiFi signal

Name	Description
Epoch time (s)	Timestamp that represents local time in AP when message was sent
Signal age (s)	Time since the last packet was heard from this station
Data rate (Mb/s)	Data rate of chirp frame
Client type (boolean)	The signal source (Either AP or device)
Channel (number)	Channel of tag transmission (Either 5 GHz or 2.4 GHz)
AP MAC (string)	MAC address of AP
Association status (boolean)	The signal data is probe or data package
RSSI (dBm)	The value of the RSSI
Noise floor (dBm)	Noise floor of the radio
Radio BSSID (string)	BSSID of the radio that detected the device
Mon BSSID (string)	BSSID of the AP that the station is associated to
Client MAC (string)	MAC address of station



(a) AP: 1 (b) AP: 2 (c) AP: 3

Figure 4: Curve fitting results for 3 different APs

4.2 Building fingerprint database

Raw RSSI data is obtained from the AP controllers, which has an access to all the APs in the building. The list of attributes that makes each signal data point is shown in the table 1. Out of all the signal attributes, we utilize epoch time, RSSI value, media access control (MAC) address of the connected AP and hashed MAC address of the connected device.

To estimate location from the RSS data it is necessary to have a way to map incoming raw data to location(s). There are 2 main techniques to obtain locations using RSS data: to build a fingerprint database or building a model using fitting techniques. Modelling attempts based on ground truth trace data are done using curve fitting and it can be seen that it's very challenging to find a pattern in the data. Graphs of curve fitting results are shown for 3 different APs in the Figure 4. Since, modelling techniques achieve low accuracy, since it is impossible to model signal to distance due to environmental constraints, we use the fingerprint based technique. Also, it helps us to take into account many cases with its corresponding signal vectors.

RSSI / AP	-55	-56	-57
1	[2, 4, 44, 60…	[94, 26, 99…	[43, 5, 18…
2	[121, 33, 11…	[3, 71, 95…	[27, 57…
3	[1, 12, 33…	[88, 13, 12…	[5, 18, 67…

Figure 5: A snapshot of the fingerprint database

Each signal data is a sequence of tuples of the form $(T_i, AP_i, RSSI_i)$. To accomplish the mapping, we have developed an Android application $TraceTracker$, which can be used simply by pinpointing the locations on the screen while walking as shown in Figure 6. The application

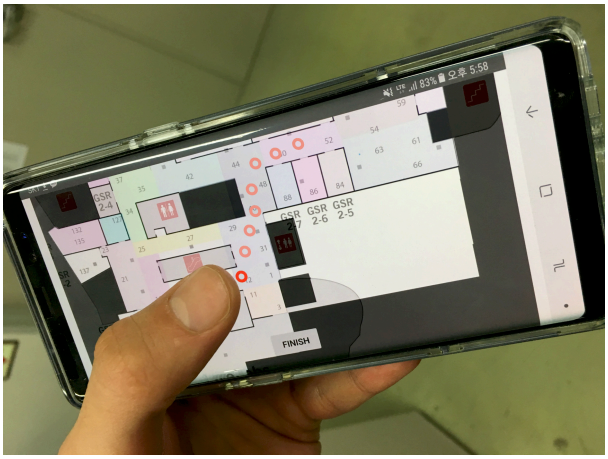


Figure 6: Android application: $TrackeTracker$

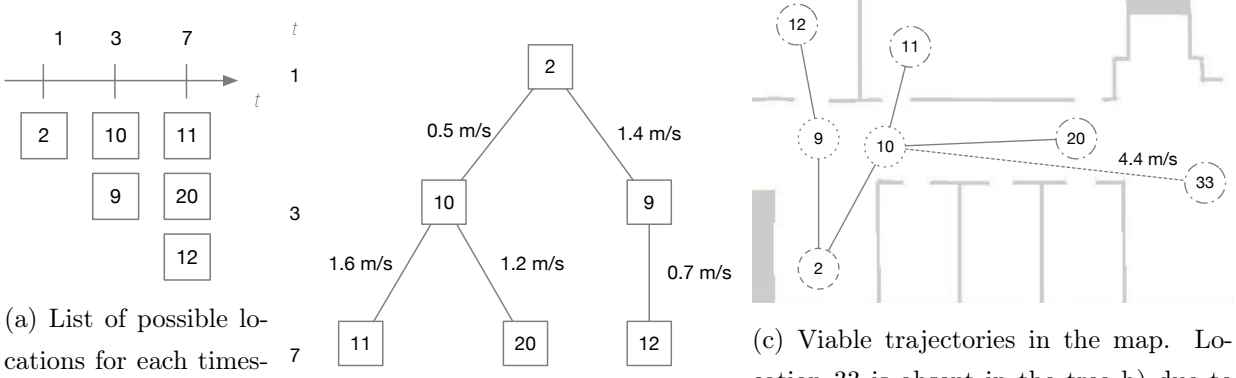
returns the list of tuples of the form (T_i, x_i, y_i), which corresponds to the time and coordinate of the points clicked on the screen during the collection process. Then, then the system matches the timestamps of the real coordinates data with the raw signal data to start creating the mapping matrix. Since, radio frequencies of 5 GHz and 2.4 GHz have different signal properties, we create 2 different matrices for each.

We assume this fingerprint collection process is considerably simple and can be done by any person familiar with the floor plan. Also, the process is more 'realistic' in the sense that during location estimation process the signals we receive would be from moving objects, not always static. However, most of the fingerprint-based solutions collect fingerprint data in a static way, which might differ from the online phase signal propagation.

Using the data, we build a matrix M which maps each historical $AP \times RSSI$ to a vector of corresponding locations (nodes) $\vec{P}$.

After these steps, we will have the fingerprint database which stores corresponding locations for tuples of the form $AP \times RSSI$, which were recorded in the ground truth collection steps using $TraceTracker$ application. The matrix have the form show in Figure 5.

V Online phase



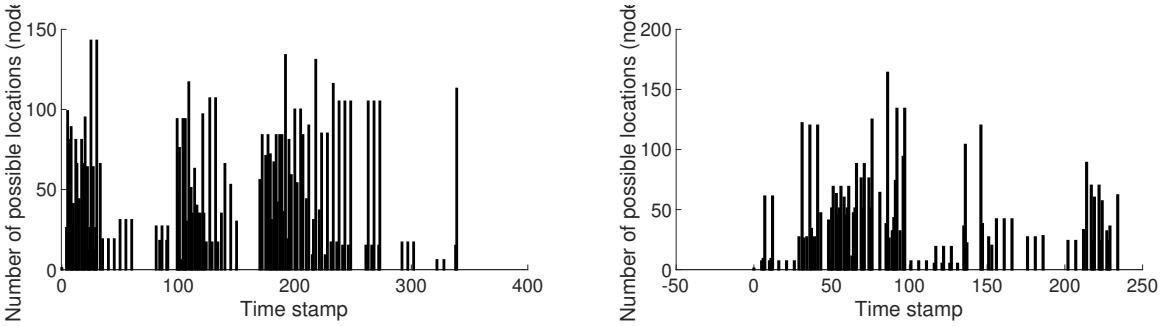
(a) List of possible locations for each timestamp

(b) Tree of all viable trajectories

(c) Viable trajectories in the map. Location 33 is absent in the tree b) due to high speed of 4.4 m/s to pass the shortest route.

Figure 7: An illustration of the *PathTree* construction

5.1 Retrieve locations from RSS data



(a) Sample trace 1

(b) Sample trace 2

Figure 8: Number of possible locations for each timestamp

The online phase of the system starts as soon as we receive raw RSS data with the attributes shown in the Table 1. In the server, we continuously receive these data and estimate the locations of users. We use the hashed MAC of the client device to distinguish the users. The input data is a sequence of tuples $(AP_i, RSSI_i, T_i)$ and the sequence is merged by T_i, so that at each t_i there is a set of records $AP_t, RSSI_t$. Then, the input will be fed into the fingerprint database built in the section 4.2 to get corresponding location estimates L_t as shown in the Algorithm 2. Now, for each time, we have a set of possible locations in the graph. As in the 4.2, the fingerprint database was built for both radio frequencies separately, which can be either 5 GHz or 2.4 GHz. The radio frequency of the signal can be identified by the attribute *channel* of from the Table 1. Our evaluations showed that localization performances of both frequencies are almost the same.

However, the fingerprint database collected out of 21 traces with around 5 minutes of walking. This database has 2.1 times more of signals in the 2.4 GHz channel than 5 GHz ones.

Algorithm 2 Location Estimator

$\overrightarrow{R}$: sequence of RSSI vectors, $\overrightarrow{R_i}$: sequence of RSSI vectors at timestamp i
$\overrightarrow{H_x}$: set of RSSI vectors corresponding to location x in the fingerprint DB
$\overrightarrow{X}$: set of locations, x: a location
$\widehat{H_x}$: divergence factor of RSS vectors at given location x
$GetEuclideanDistance(\overrightarrow{A}, \overrightarrow{B})$: a function to calculate pairwise Euclidean distance between two vectors $\overrightarrow{A}$ and $\overrightarrow{B}$
$GetMedian(\overrightarrow{A})$: returns median value of numbers in $\overrightarrow{A}$
$Append(\overrightarrow{A}, a)$: appends a to a vector $\overrightarrow{A}$
procedure ESTIMATOR(R)
    $\overrightarrow{P} \leftarrow \emptyset$
    for $\overrightarrow{R_i} \in \overrightarrow{R}$ **do** ▷ Loop sequence of RSSI vectors
        for $x \in \overrightarrow{X}$ **do** ▷ Loop through every position (grid)
            $\overrightarrow{H_x} \leftarrow \{\overrightarrow{H} | M(\overrightarrow{H}) = x\}$ ▷ M - fingerprint database
            if $\overrightarrow{H_x} = \emptyset$ **then**
                continue
            end if
            $D \leftarrow GetEuclideanDistance(\overrightarrow{R_i}, \overrightarrow{H_x})$
            if $D \leq \widehat{H_x}$ **then**
                $\overrightarrow{P_i} \leftarrow Append(\overrightarrow{P_i}, x)$
            end if
        end for
    end for
    return $\overrightarrow{P}$ ▷ Sequence of plausible locations
end procedure

After the Algorithm 2 is done running, we have a list of possible locations (nodes) for each timestamp. Since, our goal is to reduce to point-wise location estimates based on the human walking speed, we keep all the probable location points until the end. The possible locations count for traces are shown in Figure 8, where random 2 traces are chosen in Figure 8a and Figure 8b, each with around 5 minutes of walking trace.

5.2 Minimum required speed

In this subsection, we start to apply realistic trajectory estimation to improve the localization accuracy. Since, raw data might consist of invalid RSSI values or values which are not present in the fingerprint database for some of the time instances, which would make it hard to construct

realistic paths. We identify the problematic points by taking the fastest possible path between consecutive signal points. Since, each signal point was converted to a list of possible locations from the previous subsection 5.1, the minimum required speed to pass the consecutive signal points is the closest distance between consecutive list of locations divided to a duration it takes to pass the points.

Multiverse achieves preprocessing by running the following steps. by performing the following steps: *(i)* Merge the possible nodes of signals with the same timestamp *(ii)* Skip signal data if impossible to connect with its temporally neighbouring signal data, if all of the connections between consecutive location nodes exceed maximum walking speed of an average human.

5.3 $PathTree$ construction

After dropping the noisy points, we start to construct a tree of viable paths user might have taken. We call this tree of viable paths - $PathTree$. First, assuming we know the starting point of the trajectory, so at $t = 0$, there is a single possible location. Starting from the next timestamp, we keep connecting the all the possible permutations between locations of current t and previous time $t-1$. However, an important point is to construct path between two locations p_i and p_j only if the required speed to pass the shortest path between them do not exceed 4 m/s, which is already a very walking speed. Illustration of the idea is shown in the Figure 7, where set of possible locations (Figure 7a) are utilized to construct the $PathTree$ (Figure 7b), which results in viable trajectories (Figure 7c). Example of impossible route is show in the Figure 7c, where node 33 is absent in the tree Figure 7b. It is due to a high speed (4.4 m/s) requirement to pass the given route. The constructor's algorithm is fully described in the Algorithm 3 and we call the function PathTree Constructor.

5.4 $PathTree$ compressor

The $PathTree$ gets exponentially large due to high number of permutations generated from possible locations space. To avoid the exponential size, the method called $PathTree$ compressor is developed, which runs periodically at certain compression intervals and steps are thoroughly described in the Algorithm 4. In our experiments, the compression interval is set to $5seconds$. This is an important step which serves a crucial role of pruning out impossible paths and keeping the system performance at a high level by decreasing the size of the $PathTree$. It achieves it by utilizing the two key functions $SpeedConsistencyFilter()$ and $Cluster()$. $SpeedConsistencyFilter()$ is a function that prunes out the paths which have inconsistent speeds, by calculating standard deviation of the speed vector. And, $Cluster()$ function utilizes $DBScan$ method to cluster the start and ending points of path, if they are with ϵ distance, based on simple Euclidean distance calculation. In our experiments, we set the value of ϵ to 0.1.

The compressor results can be seen on the Figure 9.

Algorithm 3 PathTree Constructor

$\overrightarrow{P}$: sequence of plausible positions, $\overrightarrow{P_j}$: sequence of plausible positions at time index j,
T: tree of viable paths (PathTree), T_k: set of locations at the k_{th} level of the PathTree
τ_i: epoch time value at time index i
x_i: location at time index i
T_{min}: minimum number of nodes at any level of the PathTree
$Append(\overrightarrow{A}, a)$: appends a to a vector $\overrightarrow{A}$
$GetCount(T_k)$: number of locations at the k_{th} level of the PathTree
$GetDepth(T)$: returns depth of the PathTree T
$GetShortestDistance(x, y)$: returns the shortest path distance between locations x an y given environmental constraints
procedure CONSTRUCTOR(P)
    $T \leftarrow Tree()$ ▷ Initialization of the PathTree
    $T_1 \leftarrow \overrightarrow{P_1}$
    $k \leftarrow 1$
    for $i \in \{2, \ldots, I\}$ **do** ▷ I: last time index
        $k \leftarrow k + 1$
        for $j \in \{i, \ldots, min(i + \Delta i_{max} - 1, I\}$ **do**
            $\Delta\tau \leftarrow \tau_j - \tau_i$ ▷ Duration (s)
            for $x_i \in \overrightarrow{P_j}$ **do**
                $d \leftarrow GetShortestDistance(x_j,\ x_i)$ ▷ Distance (m)
                $v \leftarrow d/\Delta\tau$ ▷ Speed of walk (m/s)
                if $v \leq v_{max}$ **then**
                    $T_k \leftarrow Append(T_k, x_j)$
                end if
            end for
        end for
        if $GetCount(T_k) \geq T_{min}$ **then** ▷ Number of nodes in the next level $\geq T_{min}$
            break
        end if
        if $GetDepth(T) \geq T_{max}$ **then** ▷ PathTree depth $\geq T_{max}$
            $Compressor(T)$ ▷ Refer to the Algorithm 4
        end if
    end for
    return T ▷ Tree of plausible paths
end procedure

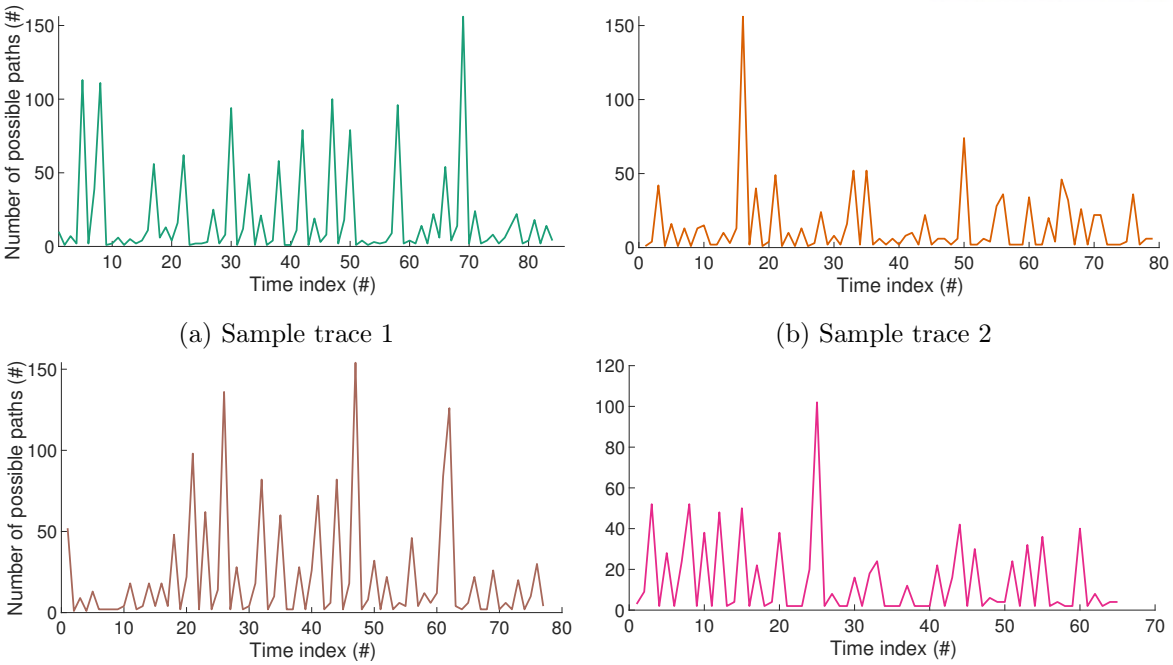
VI Implementation

Algorithm 4 PathTree Compressor

T: PathTree
L_t: list of paths from time t, N: number of timestamps of each path in L_t
$AppendChild(T, t, L)$: appends a child L_N to the node L_1 at the level t of the PathTree T and returns modified T
$Cluster(L, \epsilon)$: clusters set of paths based on start and end locations using DBScan algorithm with given parameter ϵ
$GetAllPath(T, t)$: returns list of paths of the PathTree T starting from timestamp t till now
$GetShortestPath(x, y)$: returns the shortest path between locations x an y given environmental constraints
$GetUnique(\overrightarrow{A})$: returns unique elements of vector A
$PruneTree(T, t)$: remove all the nodes of the PathTree T starting from timestamp t
$SpeedConsistencyFilter(L_t)$: filters list of paths by checking speed consistency
procedure COMPRESSOR(T)
    $L_t, N \leftarrow GetAllPath(T, t)$
    $L'_t \leftarrow SpeedConsistencyFilter(L_t)$ ▷ Prune paths with inconsistent speed sequence
    $P_{start} \leftarrow GetUnique(L'_{t,1})$
    $P_{end} \leftarrow GetUnique(L'_{t,N})$
    $\overrightarrow{L_{dirrect}} \leftarrow GetShortestPath(P_{start}, P_{end})$
    $\overrightarrow{L_{dirrect}}' \leftarrow Cluster(L_{dirrect}, \epsilon)$ ▷ Cluster direct paths using DBScan method
    $T' \leftarrow PruneTree(T, t)$
    for $L'_{direct} \in \overrightarrow{L_{dirrect}}'$ **do**
        $T' \leftarrow AppendChild(T', t, L'_{direct})$
    end for
    return T' ▷ Compressed tree of plausible paths
end procedure

We implemented Multiverse using the RSSI data feed from the Cisco Wireless LAN controller (Cisco WLC 8500), which is designed to send information about clients devices that are heard by the network in the campus of Ulsan National Institute of Science and Technology. We have developed a framework that can access to the controller at the interval of 4 seconds and dumps signal information with data fields described in the Table 1 of selected devices. We distinguish devices using their MAC address. Out of all the signal attributes, we utilize information related to each nearby AP, RSSI and its corresponding timestamp when the signal was heard. Note: every device that connects to the WiFi infrastructure at each venue has agreed to this tracking as part of the sign-on agreement. In addition, all devices that are treated as hashed entities with no additional knowledge about them.

RSSI data feed is collected in our main server and all the WiFi signal data are sent to a program written in MATLAB to send back the location information in batches Figure 10. Each



(a) Sample trace 1

(b) Sample trace 2

(c) Sample trace 3

(d) Sample trace 4

Figure 9: The effect of the PathTree Compressor on number of viable paths.

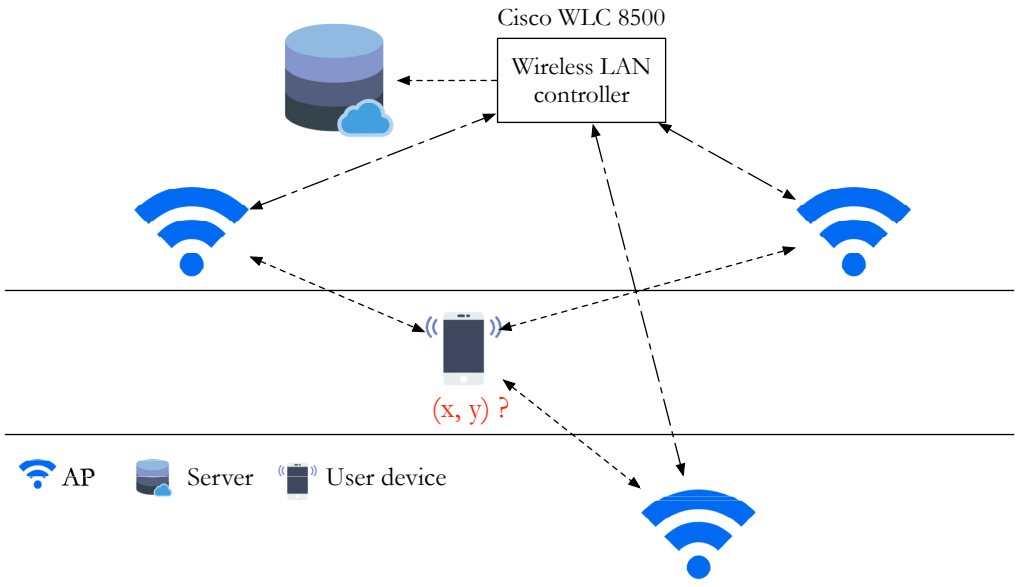


Figure 10: Implementation of the server. RSSI data is sent continuously to the main server in batches of 5 minutes.

batch is each 5 minutes to get the trace trajectory data for each device.

We used Cisco's Wireless controller device to access real-time data of users connected to

VII Evaluation

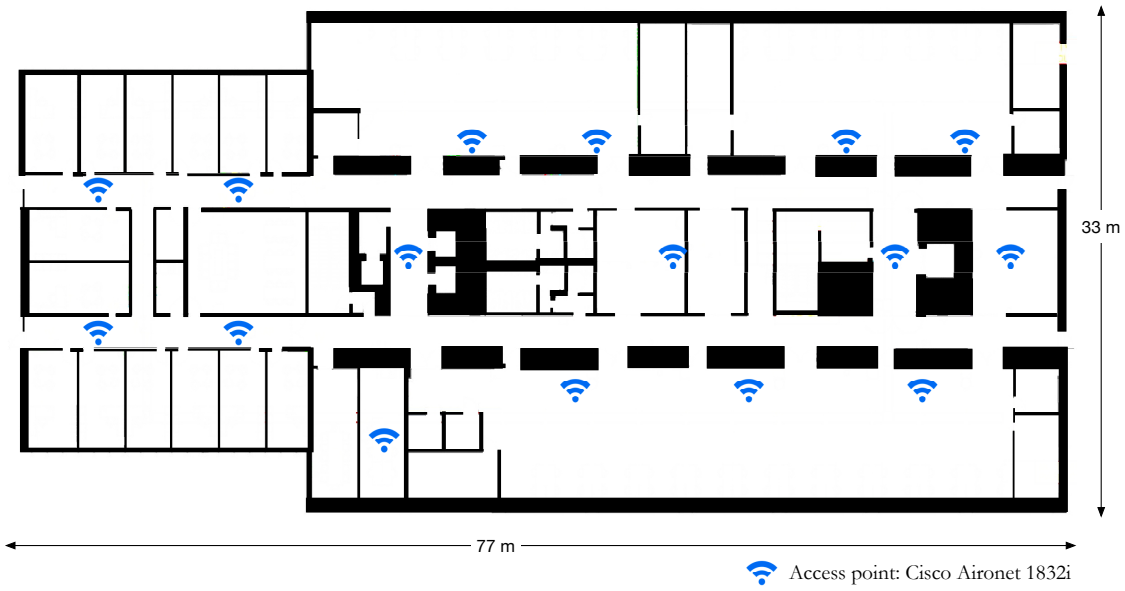


Figure 11: Floor plan of the experimentation area

7.1 Methodology

We design real-life experiments in a campus building with 16 APs installed at known locations Figure 11. Users walked around arbitrarily in the building for an hour during normal office hours, with the smartphone at their hands by clicking their current locations using the *TraceTracker*, covering approximetly 2000 m^2. As each user walks, the AP signal strength data is sent from WiFi LAN Controller to the main server. At the same time, real coordinates of the users is uploaded in the server as well from *TraceTracker*. The distance between the ground truth and the estimated locations is Multiverse's instantaneous localization error. Since, the raw data might not always correctly match temporally with the ground truth data points, we used interpolation between locations by correctly finding the midpoints between consecutive graph nodes using the shortest distances matrix built in the 4.1.

7.2 Localization error

Trace comparisons of ground truth vs Multiverse and ground truth vs Landmark-based methods are shown in the Figure 14, Figure 15 and Figure 16 for random sample traces with around 5 minutes of walking. We can most of the time find out the room-level location, and corridors are mostly easiest due to simplicity of the environment.

Our experiments show that Multiverse achieves a median localization error of 1.6 m, and the 80^{th} percentile error is 4.1 m.

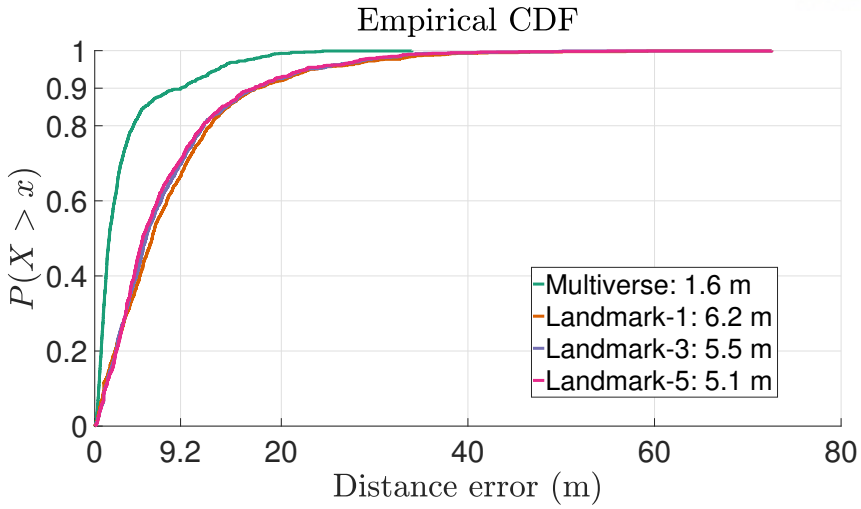


Figure 12: CDF of point-wise distance errors

Two metrics are designed for localization performance: location error and room error. Location error is defined as the Euclidean distance from the estimated location to the ground truth one. As the final outputs of Multiverse, the RSS noises and mapping errors are simultaneously taken into account. Each query contains a fingerprint and LiFS returns an estimated location. We also implemented RADAR-inspired solution [18] with 3 different configurations. Configurations are different in the way K nearest neighbors algorithm uses the value of K. Then compare their performance with Multiverse on the same experiment data. The average point-wise distance error of Multiverse is 1.6 meters, which is around 30 % smaller than Landmark-5 (5.1 meters) as can be seen in Figure 12. The performance of Multiverse is considerably better than the state-of-the-art model-based approaches (larger than 5 meters) reported in [31] and EZ (larger than 7 meters) [6]. Some location errors are caused by the symmetric structure of rooms, but they are relatively small and will not contribute to room error. This accuracy is impressive, considering Multiverse needs no site survey and no specific infrastructure.

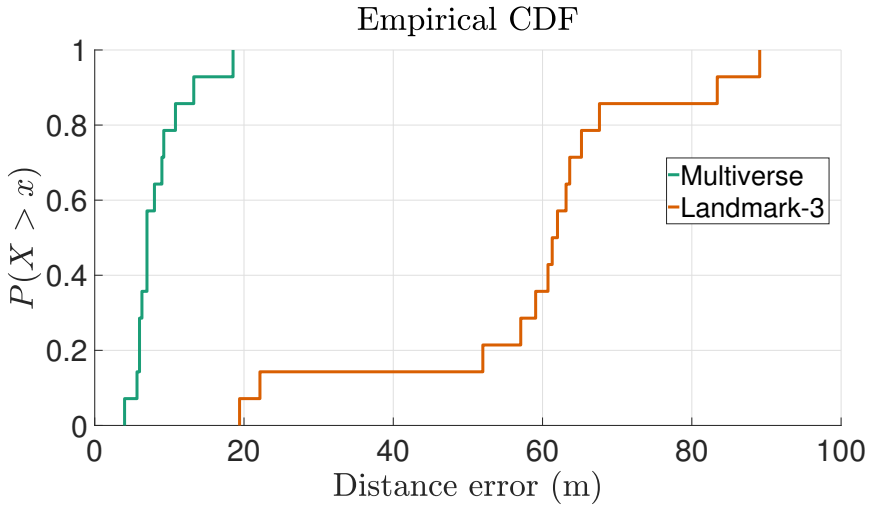
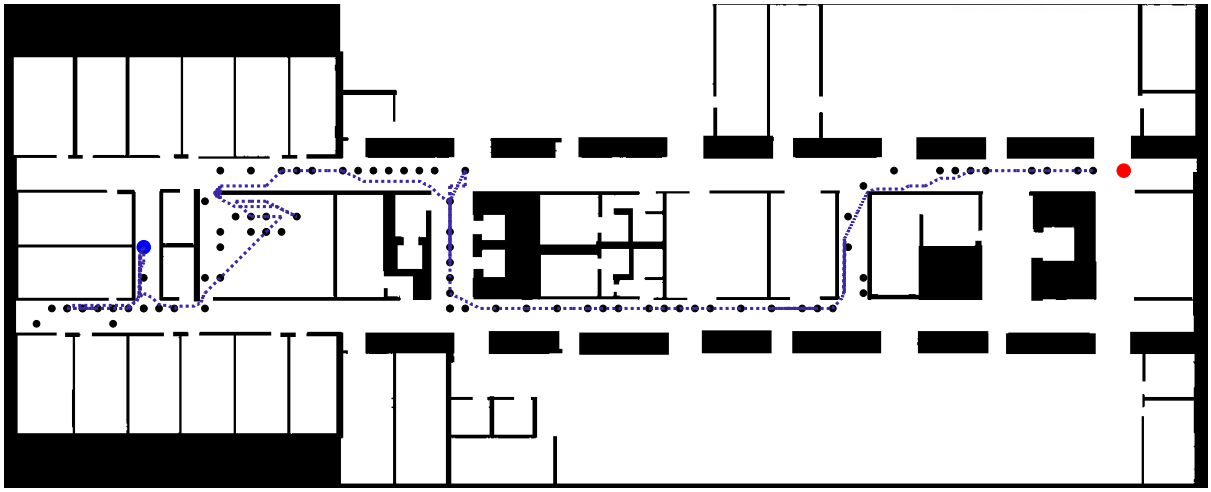
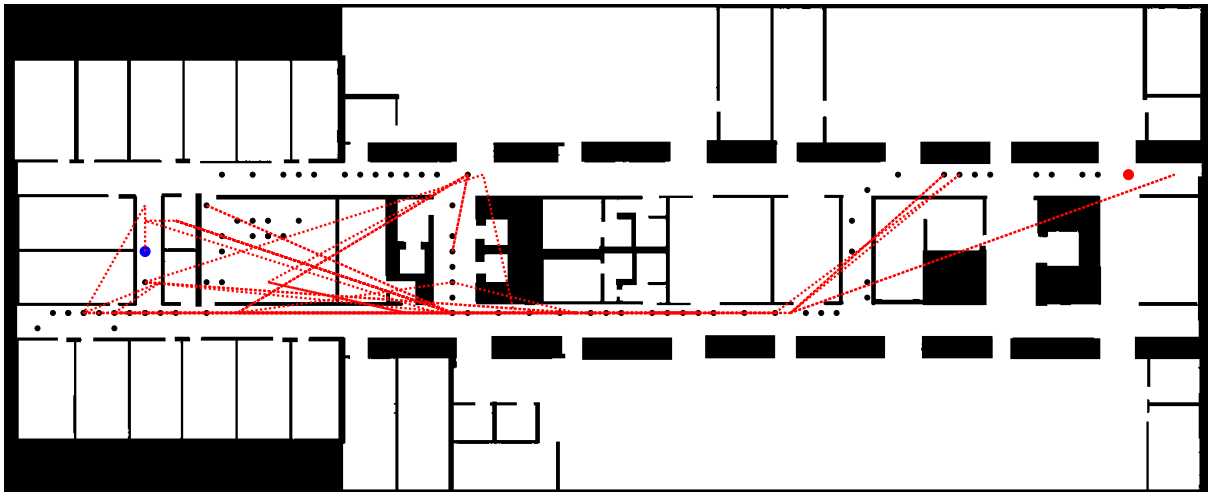


Figure 13: CDF of Fréchet distance errors



(a) Multiverse: Point wise-error: 2.5 m, Fréchet distance error: 9.2 m



(b) Landmark-based: Point wise-error: 5.5 m, Fréchet distance error: 65.2 m

Figure 14: Evaluation of sample trace 1. Multiverse trajectory (dotted blue lines) vs Landmark-based (dotted red lines) vs ground truth trace (black dots). Blue, red dots are starting and ending positions, correspondingly.

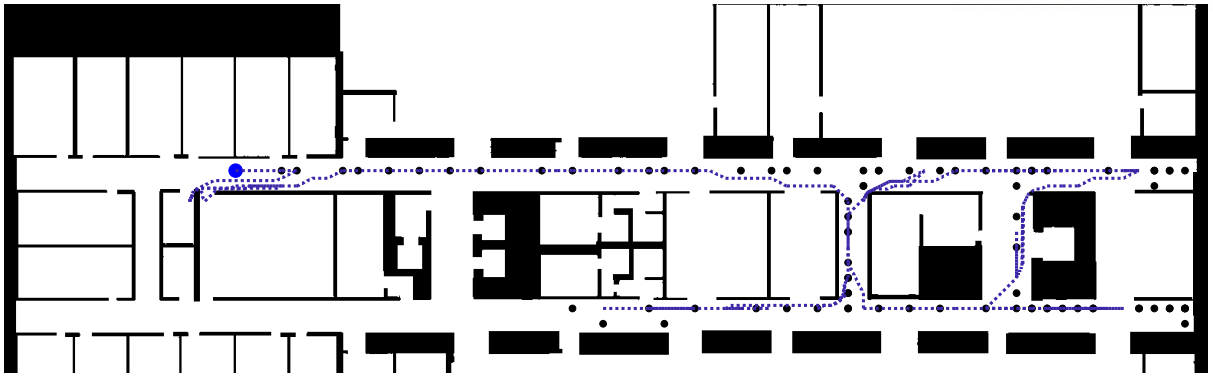
7.3 Fréchet distance error

Since Multiverse outputs trace, it is also important to ensure curve similarity. For that purpose, we implement Fréchet distance to measure ground truth curve and Multiverse output curve.

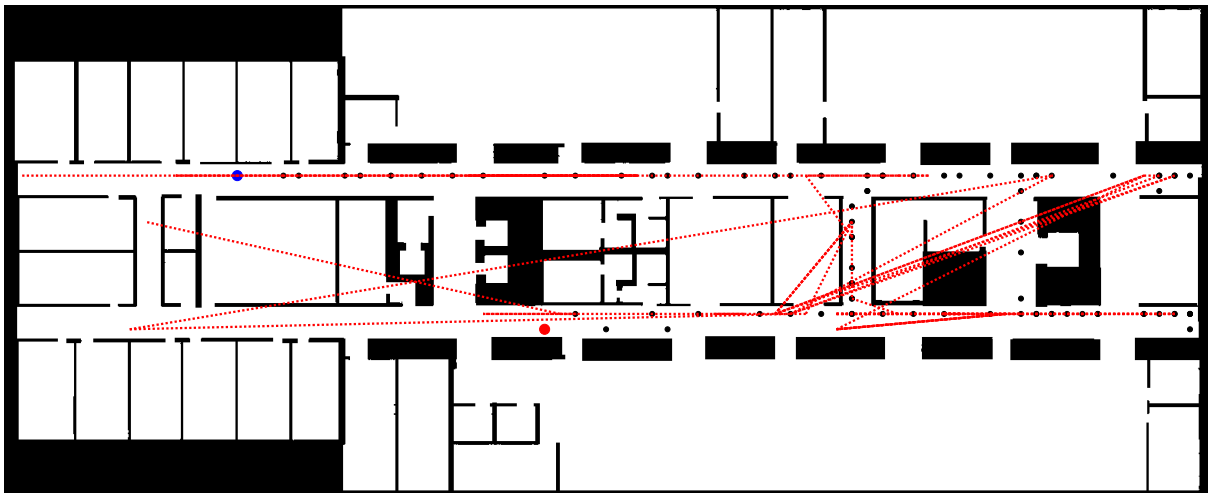
$$\delta_F(f,g) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \|f(\alpha(t)) - g(\beta(t))\| \tag{1}$$

where f and g are the two shapes and α and β are the two parameterizations and $\delta_F(f,g)$ is the Fréchet distance during time $t \in [0,1]$.

The CDF of the Fréchet distance errors are depicted in Figure 13. We compare the results of Multiverse against Landmark-3. Since, our method takes into consideration environmental constraints and uses shortest paths between consecutive locations, the errors are noticeably better



(a) Multiverse: Point wise-error: 1.1 m, Fréchet distance error: 7.0 m



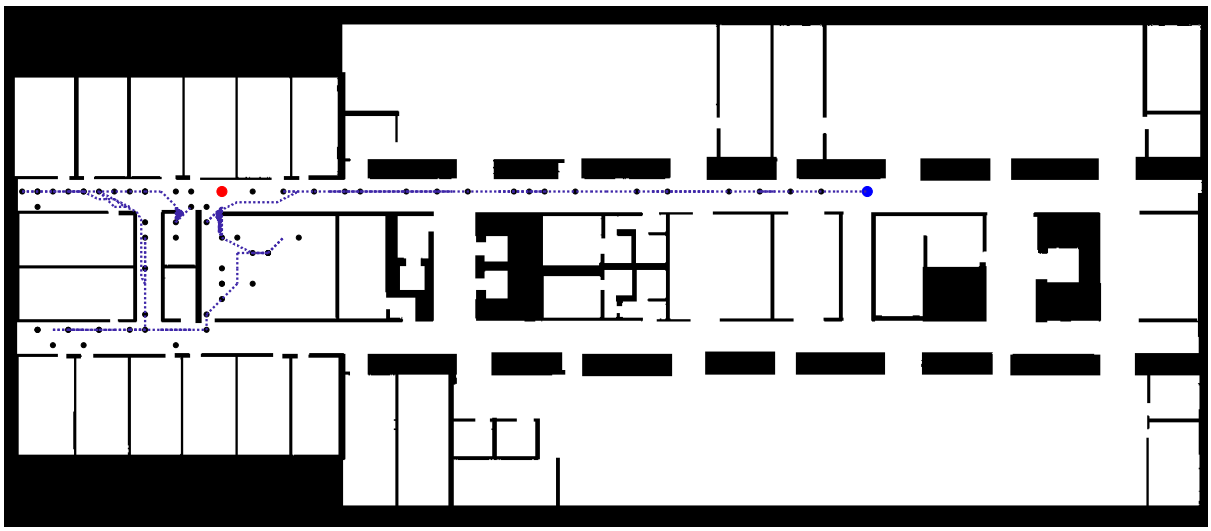
(b) Landmark-based:Point wise-error: 5.6 m, Fréchet distance error: 63.6 m

Figure 15: Evaluation of sample trace 2. Multiverse trajectory (dotted blue lines) vs Landmark-based (dotted red lines) vs ground truth trace (black dots). Blue, red dots are starting and ending positions, correspondingly.

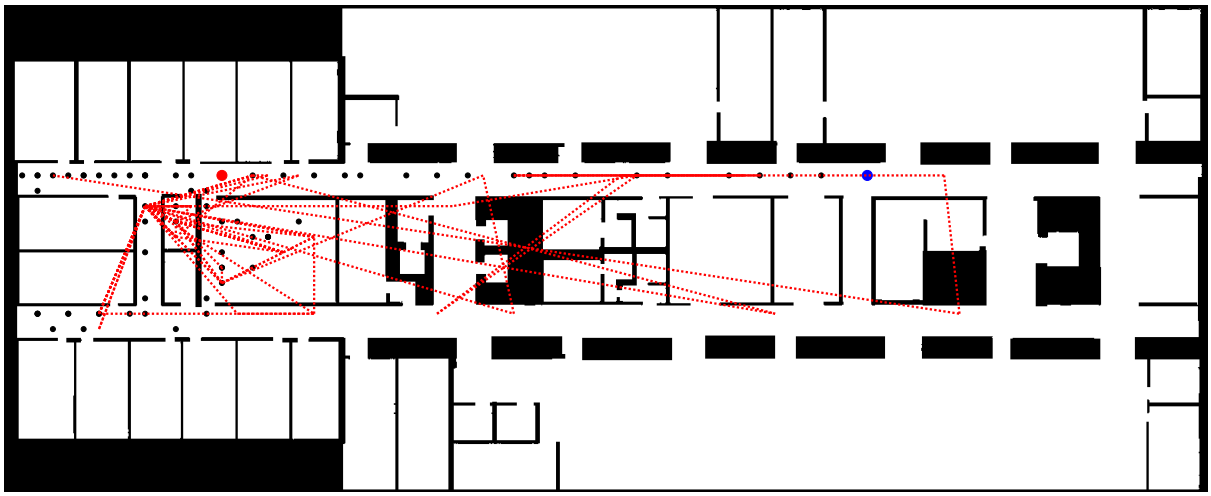
(Multiverse: [Figure 14a, Figure 15a, Figure 16a] vs Landmark-based: [Figure 14b, Figure 15b, Figure 16b]).

7.4 System efficiency

To output a estimated trace for a 5 minute trace, it takes about 1 minute of computational time in a modern CPU. The running machine has the following capabilities. Processor: P3.5 GHz Intel Core i7. Memeory: 20 GB 1600 MHz DDR3.



(a) Multiverse: Point wise-error: 2.85 m, Fréchet distance error: 8.0 m



(b) Landmark-based:Point wise-error: 7.4 m, Fréchet distance error: 62.0 m

Figure 16: Evaluation of sample trace 3. Multiverse trajectory (dotted blue lines) vs Landmark-based (dotted red lines) vs ground truth trace (black dots). Blue, red dots are starting and ending positions, correspondingly.

VIII Discussion

Most indoor localization systems rely on sophisticated signal information or device sensor data to accurately estimate the user location and traces. While Multiverse utilizes contextual information such as previous possible location and human mobility pattern information.

This work also motivates indoor positioning systems to consider curve similarity, since it's an important metrics to validate the system. It also ensures high quality analysis results such as human walking patterns in commercial buildings.

However, the system has some limitations. The location information is usually available after

certain delay due to higher than average computational demand and device heterogeneity is not guaranteed, since in the evaluations we have not utilized all the brands of smartphones with diverse WiFi hardware. It has been discussed that devices based on WiFi chipset model, might have different signal propagation patterns [41], which results in RSS vectors mismatch to bring localization accuracy down. These issues are extensively discussed and addressed in academia. For example, [42] finds that the robustness could be achieved by utilizing relative values of RSS vectors among different APs rather than absolute values. Also, [43] tries to mitigate the issue by applying kernel estimation to the signal propagation models of devices with different WiFi chipsets. While, [44] learns the features of RSS vectors and linear dependency among them to solve the robustness issue. All these attempts show encouraging results and could be applied in our method as well to achieve better robustness among devices with diverse hardware configurations.

IX Conclusion

WiFi-based indoor localization technique was developed, which can achieve practical accuracy, easily deployable and utilizes only existing WiFi infrastructure without requiring any access to data from mobile devices. All these features of the system allows large scale development for ubiquitous usage.

Impressive results achieved with already available signal information give hope to new methods that can utilize unconventional data such as human mobility patterns. In this paper, by utilizing average speed of walking and environmental constraints, we have achieved to improve localization accuracy by more than 30% comparing to the state-of-art methods relying on same information from WiFi infrastructure. Hence, we believe that gaining more insights on how humans walk give environmental constraints, the systems can achieve greater performances without disturbing their natural behaviours.

References

[1] P. Bahl and V. N. Padmanabhan, "Radar: An in-building rf-based user location and tracking system," in *INFOCOM*. IEEE, 2000.

[2] S. Liu, Y. Jiang, and A. Striegel, "Face-to-face proximity estimationusing bluetooth on smartphones," *IEEE Transactions on Mobile Computing*, vol. 2, pp. 775–784, 2014.

[3] X. Zhao, Z. Xiao, A. Markham, N. Trigoni, and Y. Ren, "Does btle measure up against wifi? a comparison of indoor location performance," in *European Wireless*. VDE, 2014, pp. 1–6.

[4] Y. Chen, D. Lymberopoulos, J. Liu, and B. Priyantha, "Fm-based indoor localization," in *ACM MobiSys*, 2012.

[5] S. Yoon, K. Lee, and I. Rhee, "Fm-based indoor localization via automatic fingerprint db construction and matching," in *ACM MobiSys*, 2013.

[6] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil, "Landmarc: indoor location sensing using active rfid," *Wireless networks*, vol. 10, no. 6, pp. 701–710, 2004.

[7] J. Wang and D. Katabi, "Dude, where's my card?: Rfid positioning that works with multipath and non-line of sight," *ACM SIGCOMM*, vol. 43, no. 4, 2013.

[8] L. Yang, Y. Chen, X.-Y. Li, C. Xiao, M. Li, and Y. Liu, "Tagoram: Real-time tracking of mobile rfid tags to high precision using cots devices," in *ACM MobiCom*, 2014.

[9] W. Zhuo, B. Zhang, S. G. Chan, and E. Y. Chang, "Error modeling and estimation fusion for indoor localization," in *IEEE ICME*, 2012.

[10] Z. Sun, A. Purohit, K. Chen, S. Pan, T. Pering, and P. Zhang, "Pandaa: physical arrangement detection of networked devices through ambient-sound awareness," in *ACM UbiComp*, 2011.

[11] W. Huang, Y. Xiong, X.-Y. Li, H. Lin, X. Mao, P. Yang, and Y. Liu, "Shake and walk: Acoustic direction finding and fine-grained indoor localization using smartphones," in *IEEE INFOCOM*, 2014.

[12] Y.-S. Kuo, P. Pannuto, K.-J. Hsiao, and P. Dutta, "Luxapose: Indoor positioning with mobile phones and visible light," in *ACM MobiCom*, 2014.

[13] Z. Yang, Z. Wang, J. Zhang, C. Huang, and Q. Zhang, "Wearables can afford: Light-weight indoor positioning with visible light," in *ACM MobiSys*, 2015.

[14] J. Chung, M. Donahoe, C. Schmandt, I.-J. Kim, P. Razavai, and M. Wiseman, "Indoor location sensing using geo-magnetism," in *ACM MobiSys*, 2011.

[15] H. Xie, T. Gu, X. Tao, H. Ye, and J. Lv, "Maloc: A practical magnetic fingerprinting approach to indoor localization using smartphones," in *ACM UbiComp*, 2014.

[16] J. Xiong and K. Jamieson, "Arraytrack: A fine-grained indoor location system," in *USENIX NSDI*, 2013.

[17] S. Kumar, S. Gil, D. Katabi, and D. Rus, "Accurate indoor localization with zero start-up cost," in *ACM MobiCom*, 2014.

[18] A. J. Khan, V. Ranjan, T.-T. Luong, R. Balan, and A. Misra, "Experiences with performance tradeoffs in practical, continuous indoor localization," in *IEEE WoWMoM*, 2013.

[19] P. Bahl, V. N. Padmanabhan, and A. Balachandran, "Enhancements to the radar user location and tracking system," *Microsoft Research*, vol. 2, no. MSR-TR-2000-12, pp. 775–784, 2000.

[20] K. Wu, J. Xiao, Y. Yi, M. Gao, and L. M. Ni, "Fila: Fine-grained indoor localization," in *IEEE INFOCOM*, 2012.

[21] K. Chintalapudi, A. Padmanabha Iyer, and V. N. Padmanabhan, "Indoor localization without the pain," in *ACM Mobicom*, 2010.

[22] B. D. Ferris, D. Fox, and N. Lawrence, "Wifi-slam using gaussian process latent variable models," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007.

[23] H. Lim, L.-C. Kung, J. Hou, and H. Luo, "Zero-configuration, robust indoor localization: Theory and experimentation," in *IEEE INFOCOM*, 2006.

[24] A. Goswami, L. E. Ortiz, and S. R. Das, "Wigem: A learning-based approach for indoor localization," in *ACM CoNEXT*, 2011.

[25] A. Kushki, K. N. Plataniotis, and A. N. Venetsanopoulos, "Kernel-based positioning in wireless local area networks," *IEEE transactions on mobile computing*, vol. 6, no. 6, 2007.

[26] J. J. Pan, J. T. Kwok, Q. Yang, and Y. Chen, "Accurate and low-cost location estimation using kernels," in *International Joint Conference on Artificial Intelligence*, 2005.

[27] S. Kumar, E. Hamed, D. Katabi, and L. Erran Li, "Lte radio analytics made easy and accessible," in *ACM SIGCOMM*, 2014.

[28] S. Sen, J. Lee, K.-H. Kim, and P. Congdon, "Avoiding multipath to revive inbuilding wifi localization," in *ACM MobiSys*, 2013.

[29] K. R. Joshi, S. S. Hong, and S. Katti, "Pinpoint: Localizing interfering radios." in *USENIX NSDI*, 2013.

[30] D. Niculescu and B. Nath, "Vor base stations for indoor 802.11 positioning," in *MobiCom*, 2004.

[31] J. Gjengset, J. Xiong, G. McPhillips, and K. Jamieson, "Phaser: Enabling phased array signal processing on commodity wifi access points," in *ACM MobiCom*, 2014.

[32] W. Sun, J. Liu, C. Wu, Z. Yang, X. Zhang, and Y. Liu, "Moloc: On distinguishing fingerprint twins," in *IEEE ICDCS*, 2013.

[33] S. Hilsenbeck, D. Bobkov, G. Schroth, R. Huitl, and E. Steinbach, "Graph-based data fusion of pedometer and wifi measurements for mobile indoor positioning," in *ACM UbiComp*, 2014.

[34] Z. Xiao, H. Wen, A. Markham, and N. Trigoni, "Lightweight map matching for indoor localisation using conditional random fields," in *IEEE IPSN*, 2014.

[35] J. Seitz, J. Jahn, J. G. Boronat, T. Vaupel, S. Meyer, and J. Thielecke, "A hidden markov model for urban navigation based on fingerprinting and pedestrian dead reckoning," in *IEEE FUSION*, 2010.

[36] A. Rai, K. K. Chintalapudi, V. N. Padmanabhan, and R. Sen, "Zee: Zero-effort crowdsourcing for indoor localization," in *ACM MobiCom*, 2012.

[37] Y. Gao, Q. Yang, G. Li, E. Y. Chang, D. Wang, C. Wang, H. Qu, P. Dong, and F. Zhang, "Xins: The anatomy of an indoor positioning and navigation architecture," in *ACM MLBS*, 2011.

[38] "Matlab image processing toolbox," 2017, the MathWorks, Natick, MA, USA.

[39] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Systems, Man, and Cybernetics Society*, vol. 9, no. 1, pp. 62–66, 1979.

[40] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[41] G. Lui, T. Gallagher, B. Li, A. G. Dempster, and C. Rizos, "Differences in rssi readings made by different wi-fi chipsets: A limitation of wlan localization," in *Localization and GNSS (ICL-GNSS)*. IEEE, 2011, pp. 53–57.

[42] J.-g. Park, D. Curtis, S. Teller, and J. Ledlie, "Implications of device diversity for organic localization," in *INFOCOM*. IEEE, 2011.

[43] A. M. Hossain, Y. Jin, W.-S. Soh, and H. N. Van, "Ssd: A robust rf location fingerprint addressing mobile devices' heterogeneity," *IEEE Transactions on Mobile Computing*, vol. 12, no. 1, pp. 65–77, 2013.

[44] L.-H. Chen, E. H.-K. Wu, M.-H. Jin, and G.-H. Chen, "Homogeneous features utilization to address the device heterogeneity problem in fingerprint localization," *IEEE Sensors Journal*, vol. 14, no. 4, pp. 998–1005, 2014.

Acknowledgements

I would first like to thank my advisor Professor Kyunghan Lee. The door to Prof. Lee's office was always open whenever I ran into a trouble spot or had a question during the development of this work or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it. I am forever indebted for his valuable lessons and generous support throughout my graduate studies.

I would also like to thank the experts who were involved in the evaluation stages of this research project: Sungyong Lee, Askar Kuvanychbekov, Moeen Mirhosseini. Without their passionate participation and input, the evaluation could not have been successfully conducted.

I would also like to acknowledge a major help from the members of our Mobile Systems & Networking lab of the Electrical Engineering and Computer science school at UNIST. Especially, the quality of work considerably improved thanks to the seniors of our lab: Junseon Kim, Sungyong Lee, Seongmin Ham. I am gratefully indebted to their very valuable comments and academic support on this thesis.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.